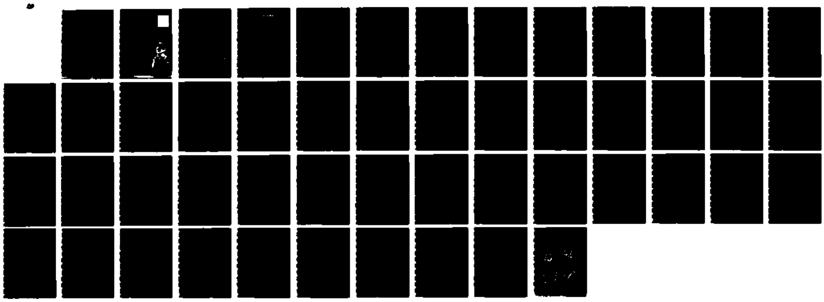
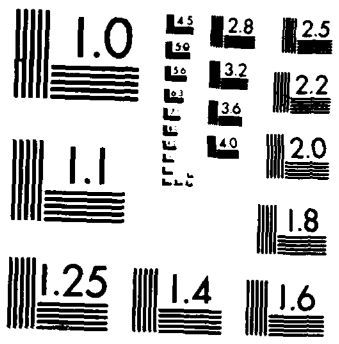
AD-A174 322 MARINE CORPS COMBAT READINESS EVALUATION SYSTEM SOFTWARE APPLICATIONS (MC (U) GEORGE WASHINGTON UNIV WASHINGTON DC INST FOR MANAGEMENT SCIE W E CAVES 1/1

UNCLASSIFIED AUG 85 GWU/IMSE/SERIAL-T-501/85 F/G 9/2 NL





CROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

(12)

Marine Corps Combat Readiness
Evaluation System
Software Applications (MCCRESSA)
Subroutine Library

by

W. E. Caves

THE
GEORGE
WASHINGTON
UNIVERSITY

STUDENTS FACULTY STUDY R
ESEARCH DEVELOPMENT FUT
URE CAREER CREATIVITY CC
MMUNITY LEADERSHIP TECH
NOLOGY FRONTIE[illegible]SIGN
ENGINEERING APP[illegible]ENC
GEORGE WASHI[illegible]NIV

[illegible]

INSTITUTE FOR MANAGEMENT
SCIENCE AND ENGINEERING

[illegible]

(12)

Marine Corps Combat Readiness Evaluation System Software Applications (MCCRESSA) Subroutine Library

by

W. E. Caves

Readiness Research
GWU/IMSE/Serial T-501/85
August 1985

THE GEORGE WASHINGTON UNIVERSITY
School of Engineering and Applied Science
Washington, DC 20052

Institute for Management Science and Engineering

Research Supported
by
Contract N00014-85-C-0716
Project NR 347 131
Office of Naval Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GWU/IMSE/Serial T-501/85	2. GOVT ACCESSION NO. AD-A174 322	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MARINE CORPS COMBAT READINESS EVALUATION SYSTEM SOFTWARE APPLICATIONS (MCCRESSA) SUBROUTINE LIBRARY		5. TYPE OF REPORT & PERIOD COVERED SCIENTIFIC
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) W. E. CAVES		8. CONTRACT OR GRANT NUMBER(s) Supported by N00014-85-C-0716
9. PERFORMING ORGANIZATION NAME AND ADDRESS GEORGE WASHINGTON UNIVERSITY INSTITUTE FOR MANAGEMENT SCIENCE AND ENGINEERING WASHINGTON, D.C. 20052		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS OFFICE OF NAVAL RESEARCH CODE 411 S&P ARLINGTON, VA 22217		12. REPORT DATE 30 August 1985
		13. NUMBER OF PAGES 44
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC SALE AND RELEASE; DISTRIBUTION UNLIMITED.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

COMBAT READINESS	TRAINING PROFICIENCY
READINESS EVALUATION	PERFORMANCE ASSESSMENT
READINESS	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A set of 24 subroutines for the Marine Corps Combat Readiness Evaluation System Software Applications (MCCRESSA) programs are documented. These subroutines, presented resident on the IBM Series/1 system at The George Washington University, are callable from either COBOL or EDL application programs operating under Version 4 of the IBM Event Driven Executive (EDX). These subroutines are written in either COBOL or the Series/1 Event Driven Executive Language (EDL) depending upon the ease/efficiency of implementing the functions performed.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

-A-

THE GEORGE WASHINGTON UNIVERSITY
School of Engineering and Applied Science
Washington, DC 20052

Institute for Management Science and Engineering

Marine Corps Combat Readiness Evaluation System Software Applications (MCCRESSA) Subroutine Library

by

W. E. Caves

Abstract
of
Readiness Research
GWU/IMSE/Serial T-501/85
30 August 1985

A set of 24 subroutines for the Marine Corps Combat Readiness Evaluation System Software Applications (MCCRESSA) programs are documented. These subroutines, presently resident on the IBM Series/1 system at The George Washington University, are callable from either COBOL or EDL application programs operating under Version 4 of the IBM Event Driven Executive (EDX). These subroutines are written in either COBOL or the Series/1 Event Driven Executive Language (EDL) depending upon the ease/efficiency of implementing the functions performed.

Research Supported
by
Contract N00014-85-C-0716
Project NR 347 131
Office of Naval Research

TABLE OF CONTENTS

0.	Introduction	1
1.	ALPHAM	3
2.	AVGPIJ	4
3.	CLEAR	6
4.	CNTSEL	7
5.	COBPGM	10
6.	DIRECT	11
7.	EVALHD	18
8.	GETALL	21
9.	GETONE	22
10.	GETWRD	23
11.	IDCODE	25
12.	IDDECODE	26
13.	PRINTL	28
14.	PUTALL	31
15.	PUTONE	32
16.	ROMNUM	33
17.	SLIDET	34
18.	SQROOT	35
19.	TRANS1	36
20.	UPCASE	38
21.	UNIQUE	39
22.	UNITI2	40
23.	UNITID	41
24.	YNRESP	42

THE GEORGE WASHINGTON UNIVERSITY
School of Engineering and Applied Science
Washington, DC 20052

Institute for Management Science and Engineering

Marine Corps Combat Readiness Evaluation System Software Applications (MCCRESSA) Subroutine Library

by

W. E. Caves

Readiness Research
GWU/IMSE/Serial T-501/85
30 August 1985

0. Introduction

The Marine Corps Combat Readiness Evaluation System (MCCRES) has been operational since July 1978 and, since that time, the Readiness Research group has performed research on MCCRES concepts, procedures, and data. Since the third quarter of Fiscal 1981, this research has been supported by an IBM Series/1 installed at The George Washington University. This support has allowed the development of MCCRES Software Applications (MCCRESSA) programs in a COBOL environment similar to that existing within Headquarters USMC (HQMC). Thus, the programs developed to support the MCCRES research conducted at The George Washington University are also suitable for operation at HQMC and, in some instances with modification, at Marine Units in the field. In fact, such MCCRESSA programs are presently operating at HQMC and in the field.

This paper describes a set of subroutines callable by either COBOL or EDL application programs. In order to facilitate its use as a reference document, each subroutine's documentation begins on a new page, is ordered alphabetically, and all pages have the subroutine's name at the top. These subroutines are available in the library volume named EDLLIB. The subroutines are made available as object modules for linking into applications programs by the data set EDLAUTO,ELDLIB. A copy of this data set appears as Figure 0.1.

References [1] through [10] contain related MCCRES material.

```
COBPGM#O,EDLLIB   COBPGM
GETONE#O,EDLLIB   GETONE
GETALL#O,EDLLIB   GETALL
PUTONE#O,EDLLIB   PUTONE
PUTALL#O,EDLLIB   PUTALL
SLIDET#O,EDLLIB   SLIDEL    SLIDER
PRINTL#O,EDLLIB   PRINTL
CLEAR#O,EDLLIB    CLEAR
ROMNUM#1,EDLLIB   ROMNUM
UNIQUE#O,EDLLIB   UNIQUE
YNRESP#O,EDLLIB   YNRESP   YN$WORK
GETWRD#O,EDLLIB   GETWRD   GW$WORK   GW$BEGIN   GW$SCANI   GW$RESPL
TRANS1#O,EDLLIB   TRANS1
UPCASE#O,EDLLIB   UPCASE#  UPCASE
ALPHAM#O,EDLLIB   ALPHAM   YN$WORK
IDCODE#O,EDLLIB   IDCODE
IDDCOD#O,EDLLIB   IDDECODE
UNITI2#1,EDLLIB   UNITI2
EVALHD#1,EDLLIB   EVALHD
SQROOT#O,EDLLIB   SQROOT    ITERCNT
DIRECT#O,EDLLIB   GETRCD    PUTRCD    CLOSEF
UNITID#O,EDLLIB   UNITID
CNTSEL#1,EDLLIB   CNTSEL
AVGPIJ#1,EDLLIB   AVGPIJ
**END
```

EDLAUTO.EDLLIB

Figure 0.1

1. ALPHAM

1.1. Description:

This EDL subroutine checks a TEXT area to determine if its contents is alphameric. Alphameric is defined here to be a string of one or more characters, each of which is either strictly alphabetic, numeric, or a member of a set of special characters, and, optionally, the first character is not numeric. In other words, the character string is a traditionally valid tag.

The results of this test is returned to the calling routine in the current length field of the TEXT area tested. If the field is strictly alphameric for the current length of the TEXT area, the current length value is unaltered. Otherwise, it is set to zero before this subroutine returns to the calling routine.

The format of the requisite EDL call statement is as follows:

```
        CALL     ALPHAM,txtadr,sctadr,ctlwrd
```

where the three arguments are defined as follows.

txtadr	The address of a TEXT area containing the word to be tested.
sctadr	The address of a TEXT area containing a set of special characters to be considered alphabetic in the alphameric test. If this argument is zero and the control word below requests that special characters be included in the test, then the three characters dollar sign ($), pound sign (#), and at sign (@) are the special characters considered alphabetic.
ctlwrd	The control word argument is passed directly. Only the two least significant bits are used by this routine. Their hex value and meaning are as follows.

HEX	Meaning
02	1 -> no special character table is to be used.
01	1 -> first character may be numeric.

1.2. External Addresses:

ALPHAM	Subroutine entry point.

1.3. Called Subroutines: None.

2. AVGPIJ

2.1. Description:

This COBOL subroutine will compute the average probability of a requirement being scored yes, P, and its standard error, SE, based upon a sample population having the number of evaluated requirements, E, and the number of those scored yes, Y. The calculations performed are defined as follows.

```
For E = 0
        P  = 0.5
        SE = 0.5
For E > 0
        For Y = 0 or Y = E
                Y = Y + 0.5
                E = E + 1.0
        P  = Y/E
        SE = ((P(1-P))/E)^(1/2)
```

$$SE = ((P(1-P))/E)^{1/2}$$

The format of the requisite COBOL call statement is as follows:

```
CALL "AVGPIJ" USING ctabi prob-est-parm
```

where ctabi is the index of the category for which the calculations are to be performed and prob-est-parm is the probability estimating category table used by FORECAST. The first parameter is simply a binary number with a COBOL definition of

```
PIC S9999 COMP SYNC.
```

The second parameter is defined in COBOL copy code named PECATPRM. This code is given in Figure 2.1. The only data elements of interest in this copy code are ECNT and YCNT, referenced by this routine, and PJ and SE, set by this routine.

2.2. External Addresses:

AVGPIJ	Subroutine entry point.

2.3. Called Subroutines: SQROOT.

```
          *                                                 02/02/85
           1   PROB-EST-PARM.
            2  STACK-CTL.
 1000        3  CAT-MAX                  PIC S9999 COMP SYNC.
    0        3  CAT-NEXT                 PIC S9999 COMP SYNC.
             3  CAT-TAB-LEV              PIC S9999 COMP SYNC.
             3  CAT-LOAD-LEV             PIC S9999 COMP SYNC.
            2  CAT-STACK.
             3  CAT-STK                  OCCURS 1000 TIMES.
              4  FROMI                   PIC S9999 COMP SYNC.
              4  NEXTI                   PIC S9999 COMP SYNC.
              4  DOWNI                   PIC S9999 COMP SYNC.
              4  CAT-STK-CTRS.
               5   RCNT                  PIC S9(9) COMP SYNC.
               5   ECNT                  PIC S9(9) COMP SYNC.
               5   YCNT                  PIC S9(9) COMP SYNC.
               5   PJ                    PIC S9(1)V9(6) COMP SYNC.
               5   SE                    PIC S9(1)V9(6) COMP SYNC.
               5   FJ                    PIC S9(1)V9(6) COMP SYNC.
              4  CAT.
               5  CAT1                   PIC X(2).
               5  CAT2                   PIC X.
               5  CAT3                   PIC X.
              4  NOMEN                   PIC X(10).
              4  CATLEV                  PIC 9.
              4  PSEFORCE                PIC X.
```

Probability Estimating Category Table

Figure 2.1

3. CLEAR

3.1. Description:

This EDL subroutine, to be called from a COBOL program, clears the static USER terminal from which the program was loaded. If the USER terminal is not static this subroutine simulates a no operation.

The format of the requisite COBOL call statement is as follows:

```
CALL "CLEAR".
```

where no arguments are required. The subroutine actually has one dummy argument which is required for any COBOL call.

3.2. External Addresses:

CLEAR	Subroutine entry point.

3.3. Called Subroutines: None.

4. CNTSEL

4.1. Description:

This COBOL subroutine will build a cross reference table of requirement counts column indexes that a) are flagged as selected, b) have an evaluation closing date that falls within a given range, and c) have a group code of interest. The selection flag table, 'from' and 'to' dates defining the evaluation closing date range, and the table of group codes of interest are typically filled by SETPRM.

The format of the requisite COBOL call statement is as follows:

```
CALL "CNTSEL" USING sel-parm crfile-hdr hdr-parm
```

where sel-parm, crfile-hdr, and hdr-parm are defined in copy code modules named SELPARM (Figure 4.1), HDRPARM (Figure 4.2), and CRHDRPRM (Figure 4.3), respectively. CNTSEL sets XREF-CNT and XREF-TAB, uses GROUP-SELECT-TEST as a work area, and references the remainder of SEL-PARM. HDR-PARM is used in the call to UNITI2. The file names and EDX volumes must be set prior to invoking this subroutine. Finally, CRFILE-HDR-PARM must have been initialized by CRINFO.

The requirement count columns scanned by CNTSEL are from OLD-NXT to NEW-CNT. If PARM-FROM-DATE is blank no date selection is made. If PARM-FROM-DATE is non-blank and PARM-TO-DATE is blank the single day defined by PARM-FROM-DATE is the date select range.

4.2. External Addresses:

CNTSEL	Subroutine entry point.

4.3. Called Subroutines: UNITI2.

```
*   REQUIREMENT FREQUENCY COUNTS FILE SELECT PARAMETER - 01/07/85
*
 1   SEL-PARM.
  2  XREF-CNT         PIC S9999 COMP SYNC.
  2  XREF-TAB         PIC S9999 COMP SYNC OCCURS 160 TIMES.
  2  PARM-FROM-DATE.
   3   MO                  PIC X(2).
   3   FILLER              PIC X.
   3   DA                  PIC X(2).
   3   FILLER              PIC X.
   3   YR                  PIC X(2).
  2  PARM-TO-DATE.
   3   MO                  PIC X(2).
   3   FILLER              PIC X.
   3   DA                  PIC X(2).
   3   FILLER              PIC X.
   3   YR                  PIC X(2).
  2  GROUP-SELECT-TEST     PIC X.
  2  GROUP-SELECT-CHAR     PIC X OCCURS 15 TIMES.
  2  EVAL-SELECT-CHAR      PIC X OCCURS 160 TIMES.
```

Requirement Counts File Select Parameter

Figure 4.1

```
*                                EVALHD/UNITI2 PARM - 01/07/85
*
 1   HDR-PARM.
  2  HP-ERR-CODES              PIC S9(9) COMP SYNC.
  2  HP-ERR-NUM                PIC S9999 COMP SYNC.
  2  HP-EVAL-ID.
   3   HP-VOL                  PIC 9(2).
   3   HP-EVAL                 PIC 9(3).
   3   HP-EVAL-DEF2 REDEFINES HP-EVAL.
    4    HP-EVAL-CHAR          PIC X OCCURS 3 TIMES.
  2  HP-UNIT-EDXVOL            PIC X(6).
  2  HP-VOL-FILE               PIC X(14).
  2  HP-LIST-FILE              PIC X(14).
  2  HP-GROUP                  PIC X.
  2  HP-SECTS.
   3   HP-SECT                 PIC X OCCURS 7 TIMES.
  2  HP-POSITIONSW             PIC X(1).
  2  HP-FIXEDSW                PIC X(1).
```

UNITI2 Parameter

Figure 4.2

```
********* COUNT REQUIREMENTS FILE HEADER MEMORY FORMAT DEFINITION
* 2149 CHARACTERS                                        11/20/84
 1   CRFILE-HDR-PARM.
  2  PARM-PART1.
   3   CAT-ID             PIC X(2).
   3   CAT-LEVEL          PIC 9.
   3   OLD-CNT            PIC 9(3).
   3   NEW-CNT            PIC 9(3).
   3   IP-GROUP-TABLE     PIC X(15).
   3   IP-VOL-TABLE       PIC X(24).
   3   IP-SECT-TABLE      PIC X(15).
  2  OLD-NXT              PIC 9(3).
  2  NEW-NXT              PIC 9(3).
  2  PARM-PART2.
   3   COUNT-ID           OCCURS 160 TIMES.
    4    ID-VOL           PIC 9(2).
    4    ID-EVAL          PIC 9(3).
    4    ID-GROUP         PIC X.
    4    ID-SECTS         PIC X(7).
```

<u>Requirement Counts File Header Parameter</u>

Figure 4.3

5. COBPGM

5.1. Description:

This EDL subroutine sets up the requisite COBOL environment and then calls a main COBOL program. The COBOL compiler normally generates three object modules which must be linked together to form a program. These object modules are identified by suffixes to the main program name as follows:

pgmnme#0	EDL entry stub for a COBOL main program.
pgmnme#1	Object code for program instructions and static data areas.
pgmnme#B	Input/Output buffer areas.

If a COBOL program is to be loaded by an EDL 'root' program the compiler supplied entry stub must be replaced. Most MCCRESSA programs are of this type and, as such, the practice has been to write a custom EDL entry stub, to name its source module 'pgmnme$S', and to name its object module 'pgmnme$0'. The common code for a custom entry stub has been assembled as this subroutine.

The format of the requisite EDL call statement is as follows:

```
CALL     COBPGM,pgmnme,cobprm,exeprm
```

where the three arguments are defined as follows.

pgmnme	The address of the entry point in the pgmnme#1 object module. This field is required.
cobprm	The address of a table of addresses locating the COBOL application program's parameter areas.
exeprm	The address a COBOL execution time parameter, if any. If no such parameter is supplied this parameter must be zero (it is presently zero in all MCCRESSA programs). If supplied, this parameter must locate a character string bounded by slashes that contains COBOL execution time option keywords.

5.2. External Addresses:

COBPGM	Subroutine entry point.

5.3. Called Subroutines: COKICIAO, COKGTRTO, & RETURN.

6. DIRECT

6.1. Description:

DIRECT#O,EDLLIB is a set of record oriented fixed block direct access input/output subroutines designed to be called from a COBOL program. The specific subroutine names are GETRCD, PUTRCD, and CLOSEF. It is assumed that any referenced file already exists, i.e., no provision is made for allocating a file, and that all physical blocks are 256 bytes. Also, file opening is implicit and file closing may or may not be implicit depending upon whether or not a call to PUTRCD was made.

6.1.1. File Control Block, First Parameter:

All input/output control areas for a file are contained within a file control block. Since the file control block is maintained within the calling routine, any number of files may be open at one time. Included within a file control block is a stack of one or more input/output buffer areas. This stack is 'rotated' with use such that a buffer's position in the stack is an indication of its 'age' in terms of last use with the top position of the stack being occupied by the most recently used buffer. No actual disk reading/writing takes place for any GETRCD or PUTRCD request if the requested logical record is already in one of the buffers. If the requested logical record is not already in one of the buffers, the 'oldest' buffer is moved to the top of the stack, the current block is written to disk only if it was modified (i.e., the target of a previous PUTRCD request), and the block containing the requested logical record is then read into the buffer. This block read does not take place if the action requested is PUTRCD and either the blocking factor is one or the requested block is 'new' to the logical file.

Figure 6.1 gives a typical file control block described in COBOL statements. For reference purposes, a line number is included at the end of each line in Figure 6.1.

6.1.1.1. The value supplied in line 02 is the decimal equivalent of hex 0104 which represents subroutine level 1 EDX level 4, the current mod-level, of DIRECT#O. This value is checked only at the first linkage to any one of the subroutines. If it is not equal to the current subroutine mod-level the subroutine issues a message and a PROGSTOP 16. After this initial program check, this area is used as a return code for the called routine. A -1 indicates that the requested action was taken. If such is not the case a return code (in the 100s) is issued.

6.1.1.2. The value supplied in line 03 is the indication that the file is not open. The calling program may, at any time, cause the next GETRCD or PUTRCD linkage to open the file by setting this field to zero. This field and the following 64 bytes are used as an EDL DSCB. The first word of the DSCB is always -1 except when the previous action was unsuccessful in which case this value is greater than zero.

```
1   IO-FILE-CONTROL-BLOCK.                                         01
 2  IO-RET-CODE        PIC S9999 COMP SYNC VALUE 260.              02
 2  IO-DSCB-CODE       PIC S9999 COMP SYNC VALUE 0.                03
 2  FILLER             PIC X(64).                                  04
 2  IO-DSN             PIC X(8) VALUE "DDDDDDDD".                  05
 2  IO-VOL             PIC X(6) VALUE "VVVVVV".                    06
 2  IO-LOG-RCD-LNG     PIC S9999 COMP SYNC VALUE n1.               07
 2  IO-STACK-SIZE      PIC S9999 COMP SYNC VALUE n2.               08
 2  IO-BLOCK-FAC       PIC S9999 COMP SYNC.                        09
 2  IO-ALLOC-SIZE      PIC S9(9) COMP SYNC.                        10
 2  IO-USED-SIZE       PIC S9(9) COMP SYNC.                        11
 2  IO-LAST-BLOCK      PIC S9(9) COMP SYNC.                        12
 2  IO-BUF-I           PIC S9999 COMP SYNC.                        13
 2  IO-DEBLK-I         PIC S9999 COMP SYNC.                        14
 2  FILLER             PIC X(2*n2).                                15
 2  FILLER             PIC X(2).                                   16
 2  IO-BUFFERS         OCCURS n2 TIMES.                            17
  3   IO-BUF-BLOCK-NO  PIC S9(9) COMP SYNC.                        18
  3   IO-BUF-MOD-SW    PIC X.                                      19
  3   FILLER           PIC X.                                      20
  3   IO-BUF-RCD       PIC X(n1) OCCURS n3 TIMES.                  21
  3   FILLER           PIC X(n4).                                  22
```

<u>File Control Block</u>

Figure 6.1

6.1.1.3. The values supplied in lines 05 and 06 are used to update the DSCB during an open operation only. An open operation takes place during a GETRCD or PUTRCD linkage with IO-DSCB-CODE zero. If the requested record number (see paragraph 6.1.2 below) is zero the linkage is simply an open request. Otherwise, the requested get/put action takes place following a successful open.

6.1.1.4. The value supplied in line 07, n1, is the logical record length for the subject file. It may be any value from 1 to 256. This n1 also appears as the length of IO-BUF-RCD in line 21.

6.1.1.5. The value supplied in line 08, n2, gives the number of buffers used for I/O operations. It may be any value greater than 0. The area reserved by line 15 must be two bytes for each buffer and the number of buffers defined in line 17 must be n2.

6.1.1.6. Each buffer contains n3 logical records of n1 bytes each as defined in line 21. Here, n3 = floor(256/n1). Any unused space is defined by the filler in line 22 where n4 = (256 - n1*n3). This line is omitted if n4 = 0.

6.1.1.7. Lines 09 through 22 are initialized during an open operation and updated as appropriate during normal subroutine operation.

6.1.1.8. IO-RET-CODE and IO-DSCB-CODE may be redefined together as PIC S9(9) COMP SYNC in which case the value would be -1 for all successful subroutine calls. If the value is not -1, IO-RET-CODE and IO-DSCB-CODE should each be analyzed to determine the error. The possible values for each of these return codes are given in paragraph 6.1.6 below.

6.1.1.9. The following is an alternative definition for lines 17 thru 22.

```
     2  FILLER            PIC X(262) OCCURS n3 TIMES.        17
```

6.1.2. Logical Record Area, Parameter two.

The linkages GETRCD and PUTRCD each require a second parameter. This parameter, again described in COBOL statements with line numbers (continued from Figure 6.1), is given in Figure 6.2. This parameter supplies the logical record number (line 32) and the logical record (line 33). Here, the n1 must be the same value as the n1 appearing on line 07.

```
    1   LOG-RCD-AREA.                                           31
     2  LOG-RCD-NUM        PIC S9(9) COMP SYNC.                 32
     2  LOG-RCD            PIC X(n1).                           33
```

<u>Logical Record Area</u>

Figure 6.2

6.1.3. Get Record Entry.

A GETRCD request may be made for any record number 0 thru IO-USED-SIZE. As stated earlier, a request with a logical record number of zero is simply an open request. A request with a record number greater than IO-USED-SIZE returns with an IO-RET-CODE of 110 indicating end-of-file and the LOG-RCD area unchanged. An actual get record request has a logical record number from 1 thru IO-USED-SIZE. Such a request searches the buffers to determine if the requested block is in memory. If the requested block is not in memory, the 'oldest' buffer is moved to the top of the stack. It is then checked to determine if the current contents have been modified. If its contents have been modified the block is written to disk before the requested block is read into the buffer. In any event, the requested logical record is moved to LOG-RCD and the IO-BUF-I and IO-DEBLK-I fields are updated.

6.1.4. Put Record Entry.

A PUTRCD request may be made for any record number 0 thru IO-USED-SIZE+1 that does not exceed IO-ALLOC-SIZE. As above, a request for logical record number 0 is simply an open request. A request with a record number greater tha IO-ALLOC-SIZE returns an IO-RET-CODE of 120 which is an output end-of-file indication. A request with a record number greater than IO-USED-SIZE+1 returns an IO-RET-CODE of 104 indicating that a 'gap' in the file would be created if the request were honored. An actual put record request has a logical record number form 1 thru IO-USED-SIZE+1. Such a request searches the buffers to determine if the requested block is in memory. If the requested block is not in memory, the 'oldest' buffer is moved to the top of the stack. It is then checked to determine if the current contents have been modified. If its contents have been modified the modified block is written to disk. In any event, the requested block is read into the buffer if its number does not exceed IO-LAST-BLOCK and the blocking factor is greater than one. If its number does exceed IO-LAST-BLOCK, IO-LAST-BLOCK is stepped. If no read takes place, the buffer block is initialized to binary zero. Finally, LOG-RCD is moved to the buffer and the IO-BUF-I and IO-DEBLK-I fields are updated.

6.1.5. Close File Entry.

A CLOSEF request need only be made if a previous PUTRCD request was issued. The CLOSEF will first write to disk any modified blocks in the buffer stack and then load $DISKUT3 to set the EOD marker.

6.1.6. Return Codes.

Following a successful linkage to one one of the DIRECT subroutines the first two words of the File Control Block will each have a value of -1 (this is also a doubleword value of -1). An unsuccessful linkage will have an IO-RET-CODE value as defined in Figure 6.3. The IO-DSCB-CODE may also have a value other than -1. If this is the case the value may be an EDX Read/Write Return Code as given in Figure 6.4 or a $DISKUT3 Return Code as given in Figure 6.5. Figures 6.4 and 6.5 were copied from the Event Driven Executive Messages and Codes Manual (SC34-0445-1). Which figure applies may be determined from the value of IO-RET-CODE.

6.2. External Addresses:

GETRCD	Read Record Subroutine entry point.
PUTRCD	Write Record Subroutine entry point.
CLOSEF	Close File Subroutine entry point.

6.3. Called Subroutines: None.

100	An open error occurred, the contents of IO-DSCB-CODE indicate the type of error. If this value is -1 then the error was that the subject data set was PGM rather than DATA..
101	A read error has occurred, the contents of IO-DSCB-CODE indicate the type of error. This error code can occur following either a GETRCD or PUTRCD request.
102	A write error has occurred, the contents of IO-DSCB-CODE indicate the type of error. This error code can occur following any linkage.
104	Record number is greater than the highest record number in the file plus one following a PUTRCD link. Honoring the request would create a gap in the file.
110	End-of-file indicator following a GETRCD link.
120	End-of-file indicator following a PUTRCD link.
190	A CLOSEF request has been made and the subject file is not open.

IO-RET-CODE Error Values

Figure 6.3

Disk and Diskette Read/Write Return Codes *(continued)*

Return Code	Condition
-1	Successful completion
1	I/O error and no device status present (this code may be caused by the I/O area starting at an odd byte address)
2	I/O error trying to read device status
3	I/O error retry count exhausted
4	Read device status I/O instruction error
5	Unrecoverable I/O error
6	Error on issuing I/O instruction for normal I/O
7	A no record found condition occurred a seek for an alternate sector was performed, and another no record found occurred, for example, no alternate is assigned
8	A system error occurred while processing an I/O request for a 1024-byte sector diskette
9	Device was offline when I/O was requested
10	Record number out of range of data set--may be an end-of-file (data set) condition
11	Data set not open or device marked unusable when I/O was requested
12	DSCB was not OPEN; DDB address = 0
13	If extended deleted record support was requested ($DCSBFLG bit 3 on), the referenced sector was not formatted at 128 bytes/sector or the request was for more than one 256-byte sector. If extended deleted record support was not requested ($DSCBFLG bit 3 off), a deleted sector was encountered during I/O.
14	The first sector of the requested record was deleted
15	The second sector of the requested record was deleted
16	The first and second sectors of the requested record were deleted

Note: The actual number of records transferred is in the second word of the TCB.

EDX Read/Write Return Codes

Figure 6.4

Return Codes (by Function)

$DISKUT3 Return Codes

The $DISKUT3 program places a return code in the first word of a data set control block specified in a DSCB statement

Return Code	Condition
1	Invalid request code parameter (not 1-6)
2	Volume does not exist (All functions)
4	Insufficient space in library (ALLOCATE)
5	Insufficient space in directory (ALLOCATE)
6	Data set already exists - smaller than the requested allocation
7	Insufficient contiguous space (ALLOCATE)
8	Disallowed data set name, eg $$EDXVOL or $$EDXLIB (all functions except OPEN)
9	Data set not found (OPEN, RELEASE, RENAME)
10	New name pointer is zero (RENAME)
11	Disk is busy (ALLOCATE, DELETE, RELEASE, RENAME)
12	I/O error writing to disk (ALLOCATE, DELETE, RELEASE, RENAME)
13	I/O error reading from disk (All functions)
14	Data set name is all blanks (ALLOCATE, RENAME)
15	Invalid size specification (ALLOCATE)
16	Invalid size specification (RELEASE)
17	Mismatched data set type (DELETE, OPEN, RELEASE, RENAME)
18	Data set already exists - larger than the requested allocation
19	SETEOD only valid for data set of type "data"
20	Load of $DISKUT3 failed ($RMU only)
21	Tape data sets are not supported
23	Volume not initialized or Basic Exchange Diskette*

* The Basic Exchange Diskette has been opened.

$DISKUT3 Return Codes

Figure 6.5

7. EVALHD

7.1. Description:

This COBOL subroutine constructs a character string text line containing selected identification information regarding a MCCRES Evaluation. The form of the requisite COBOL call statement is as follows:

```
CALL "EVALHD" USING text-line eval-parm loc-tab rtn-tab
```

where the four required parameters are defined as follows:

```
 1   text-line.                                                01
  2  FILLER           PIC S9999 COMP SYNC.                     02
  2  LENGTH           PIC S9999 COMP SYNC.                     03
  2  TEXT             PIC X(length).                           04
*                                                              05
 1   eval-parm                                                 06
  2  EP-ERR-CODES     PIC S9(9) COMP SYNC.                     08
  2  EP-ERR-NUM       PIC S9999 COMP SYNC.                     09
  2  EP-VOL           PIC 99.                                  10
  2  EP-EVAL          PIC 999.                                 11
  2  EP-UNIT-EDXVOL   PIC X(6).                                12
  2  EP-VOL-FILE      PIC X(14).                               13
  2  EP-LIST-FILE     PIC X(14).                               14
  2  EP-GROUP         PIC X.                                   15
  2  EP-SECTIONS      PIC X(7).                                16
  2  EP-POSITION-SW   PIC X.                                   17
  2  EP-FIXED-SW      PIC X.                                   18
*                                                              19
 1   loc-tab.                                                  20
  2  LOCATION         PIC S9999 COMP SYNC OCCURS n TIMES.      21
*                                                              22
 1   rtn-tab                                                   23
  2  RTN-INDEX        PIC S9999 COMP SYNC OCCURS n+1 TIMES.    24
```

The following notes refer to the interpretation of these four parameters by EVALHD and to the responsibilities of the calling routine. In order to facilitate the discussion, reference is made to the 'line numbers' assigned to the COBOL statements appearing above.

a) The length given in line 02 must be no greater than the length defined for line 03. Further, the initial contents of TEXT (line 03) are not significant.

b) Lines 06 through 12 of eval-parm are used as the first parameter when calling UNITI2. See the discussion for UNITI2 regarding these lines.

c) EP-VOL-FILE is a 14 character field where the first eight characters contain the name of the MCCRES Volume Nomenclature File and the last six characters contain the EDX volume name where this file resides.

d) EP-LIST-FILE is a 14 character field where the first eight characters contain the name of the MCCRES List Nomenclature File and the last six characters contain the EDX volume name where this file resides.

e) EP-GROUP must contain the MCCRES Group (List) code that applies to the MCCRES Evaluation identified by lines 10 and 11.

f) EP-SECTIONS may be blank or contain up to seven MCCRES Section codes which, if requested, may be included in the final contents of TEXT (line 04).

g) EP-POSITION-SW is a four valued switch which controls the final justification of the contents of TEXT (line04). The four values are as follows:

```
   N -> none
   L -> left
   C -> center
else -> right
```

h) EP-FIXED-SW is a two valued switch that controls the field width associated with each identification data element when it is placed in TEXT by this routine. If this switch has a value of 'Y', the implied length is that given as the maximum length in i) below. Otherwise, the length is the location of the last non-blank character when the first position of the field is taken to be location one.

i) Each element of rtn-tab, save the last, gives an index of the routine that processes a particular identification element. This table is scanned, processing each element in order, until a zero entry is encountered. Thus, for n elements the table must contain n+1 entries. The valid routine indexes, the corresponding identification data element, and its intrinsic maximum field length are given as follows:

1 ->	MCCRES Volume Roman Numeral	11 char
2 ->	MCCRES Volume Name Text	20
3 ->	Evaluation Number	8
4 ->	Group ID Code	7
5 ->	Included Section Codes	16
6 ->	MCCRES List Text	20
	(if unknown)	7
7 ->	MCCRES Unit ID	20
8 ->	Evaluation Closing Date	21
9 ->	MCCRES List ID	7
10 ->	MCCRES POR Reference ID	12

j) Each element of loc-tab determines the beginning position in TEXT for the identification element whose routine index is the corresponding entry in rtn-tab. Thus the dimension of loc-tab need only be n when that of rtn-tab is n+1. Values greater than zero give the beginning location in absolute terms. Otherwise, the absolute value gives the number of spaces between the present field and its predecessor. Initially the predecessor ends at location zero.

In operation, EVALHD first blanks TEXT, scans rtn-tab processing each identification data element in turn until a routine index of zero is encountered, justifies the final contents of TEXT as requested by EP-POSITION-SW, and then returns to the calling routine. Processing implies moving the subject identification data element to TEXT beginning at the column controlled by the current value of the column-index counter and the corresponding entry in loc-tab and then setting the column-index counter to locate the next available column as controlled by the value of EP-FIXED-SW, i.e., using the maximum length of the subject identification data element or its current length. No order is required for processing data elements and no check is made to protect against field overlap or TEXT overflow (characters falling outside TEXT are simply ignored).

7.2. External Addresses:

EVALHD	Subroutine entry point.

7.3. Called Subroutines: GETRCD, ROMNUM, SLIDEL, & UNITI2.

8. GETALL

8.1. Description:

This EDL subroutine, intended to be called from a COBOL program, translates hex coded information, packed two hex digits per byte, to character coded information, packed one character per byte. The primary use of this subroutine is to unpack MCCRES scores from the 80 character area in the MCCRES Evaluations Master File format to a 160 character area for processing by a COBOL program. The actual translation performed is based upon a 16 byte translate table which provides the translated character for each of the possible hex values.

The form of the requisite COBOL call statement is as follows:

```
CALL "GETALL" USING source,trans,dest,index
```

where the four arguments are defined as follows.

source	The hex coded source area which must be at least (L+1)/2 bytes long where L is the value of the fourth argument.
trans	The 16 byte translate table.
dest	The character destination area which must be at least L bytes long where L is the value of the fourth argument.
index	The index (PIC S9999 COMP SYNC) of the last hex code to be translated. Presently this must have a value of from 1 to 160, inclusive.

8.2. External Addresses:

GETALL	Subroutine entry point.

8.3. Called Subroutines: None.

9. GETONE

9.1. Description:

This EDL subroutine, intended to be called from a COBOL program, translates one hex coded digit to a character coded byte. The primary use of this subroutine is to unpack a MCCRES score from the 80 character area in the MCCRES Evaluations Master File format to a character code for processing by a COBOL program. The actual translation performed is based upon a 16 byte translate table which provides the translated character for each of the possible hex values.

The form of the requisite COBOL call statement is as follows:

```
CALL "GETONE" USING source,trans,dest,index
```

where the four arguments are defined as follows.

source	The hex coded source area which must be at least (L+1)/2 bytes long where L is the value of the fourth argument.
trans	The 16 byte translate table.
dest	The character destination byte.
index	The index (PIC 9999 COMP SYNC) of the hex code to be translated. Presently this must have a value of from 1 to 160, inclusive.

9.2. External Addresses:

GETONE	Subroutine entry point.

9.3. Called Subroutines: None.

10. GETWRD

10.1. Description:

This EDL subroutine will deliver to the calling routine a 'word' from the user. Under control of the calling routine, the word delivered may be a complete user response (excluding leading and trailing blanks but including all internal blanks), a single word from a user response (delimited by blanks), or a null user response. The 'word' returned may be either the next 'word' from a previously entered user response (no prompt given) or the first 'word' from a new user response (prompt given if supplied by the calling routine). A null user response is defined as no non-blank characters supplied by the user and is usually simply a carriage return following the prompt message. It may also arise when the request is for a word from a stored response only and no such word remains.

The format of the requisite EDL call statement is as follows:

```
        CALL      GETWRD,msgadr,wrdadr,trtadr,ctlwrd
```

where the four arguments are defined as follows.

msgadr	The address of a TEXT statement containing the prompting character string. If no prompt is desired this argument must be zero.
wrdadr	This argument is the address of either the TEXT area where the delivered word is to be placed or the area where the address of the beginning of the word is to be stored. Which form this argument takes is determined by the control word defined below.
trtadr	The address of a 256 byte translate table which is to be used to translate the user supplied word to the returned word. Normally this table simply translates lower case letters to upper case letters leaving all other bit combinations as is. This argument must be zero if no translation is desired.
ctlwrd	This argument is a control word which is divided into two parts. The first byte, undetermined by the calling routine, is set to the length of the delivered 'word' by this subroutine. The second byte consists of eight control switches which are defined as follows:

HEX	MEANING
80	(reserved - 0)
40	(reserved - 0)
20	(reserved - 0)
10	1 -> Force user response.
08	1 -> Return 'word' indirectly (argument 2 is address of 'word' address.
04	1 -> Return only stored response (no-read switch)
02	1 -> Return all of user response.
01	1 -> User response to be on line following prompt.

10.2. External Addresses:

GETWRD	Subroutine entry point.
GW$WORK	User response area (80 characters - not normally referenced)
GW$BEGIN	Index of the location in GW$WORK where the last returned word begins.
GW$SCANI	Index of the location in GW$WORK following the last return word.
GW$RESPL	Index of the location in GW$WORK of the last non-blank character of the most recent user response.

10.3. Called Subroutines: None.

11. IDCODE

11.1. Description:

This EDL subroutine, intended to be called from a COBOL program, translates a thirteen byte numeric/character count column identification field suitable for COBOL processing into six byte binary/hex coded field for inclusion in the header record of a MCCRES Requirement Counts File header record. A count column identification field identifies the MCCRES Volume, historical evaluation number, group code, and MCCRES Section selection, if any, for a Requirement Counts Column.

The form of the requisite COBOL call statement is as follows:

```
CALL "IDCODE" USING id-parm
```

where id-parm has the following COBOL definition.

```
1   id-parm.
 2  GROUP-TABLE       PIC X(16) VALUE "0123456789ABCDEF".
 2  SECTION-TABLE     PIC X(15).
 2  CODED-ID          PIC X(06).
 2  DECODED-ID.
  3   ID-VOL          PIC 99.
  3   ID-EVAL         PIC 999.
  3   ID-GROUP        PIC X.
  3   ID-SECTS        PIC X OCCURS 7 TIMES.
```

Any section code appearing in ID-SECTS must be contained in SECTION-TABLE. Similarly, ID-GROUP must be contained in GROUP-TABLE. This subroutine moves information from DECODED-ID to CODED-ID as follows:

a) The binary value of the first byte of CODED-ID is set equal to ID-VOL.

b) The binary value of the second byte of CODED-ID is set equal to ID-EVAL.

c) The hex value of the first half of the third byte of CODED-ID is set equal to the offset in GROUP-TABLE of the character in ID-GROUP.

d) The indexes of any non blank characters in ID-SECTS are stored in the seven hex values located in the second half of the third byte and the remaining three characters of CODED-ID. Blank positions are represented by hex zero.

11.2. External Addresses:

IDCODE   Subroutine entry point.

11.3. Called Subroutines: None.

12. IDDECODE

12.1. Description:

This EDL subroutine, intended to be called from a COBOL program, translates a six byte binary/hex coded count column identification field, as it appears in a MCCRES Requirement Counts File header record, into a thirteen byte numeric/character field suitable for COBOL processing. The coded count column identification field which this subroutine decodes was previously the product of the companion coding subroutine, IDCODE, which operates from a thirteen byte numeric/character field similar to the output of this subroutine. Either of these fields identify the MCCRES Volume, historical evaluation number, corresponding group code, and MCCRES Section selection, if any, for a Requirement counts column.

The form of the requisite COBOL call statement is as follows:

```
CALL "IDDECODE" USING id-parm
```

where id-parm has the following COBOL definition.

```
1   id-parm.
 2  GROUP-TABLE       PIC X(16) VALUE "0123456789ABCDEF".
 2  SECTION-TABLE     PIC X(15).
 2  CODED-ID          PIC X(06).
 2  DECODED-ID.
  3   ID-VOL          PIC 99.
  3   ID-EVAL         PIC 999.
  3   ID-GROUP        PIC X.
  3   ID-SECTS        PIC X OCCURS 7 TIMES.
```

SECTION-TABLE must have been initialized to the section table present in part one of the Count Requirements File header record and CODED-ID set to one of the six byte count identification fields in part two of the same header record. This subroutine moves information from CODED-ID to DECODED-ID as follows:

a) ID-VOL is set equal to the binary value of the first byte of CODED-ID.

b) ID-EVAL is set equal to the binary value of the second byte of CODED-ID.

c) ID-GROUP is set equal to the character in GROUP-TABLE whose offset is the hex value of the first half of the third byte of CODED-ID.

d) The second half of the third byte and the remaining three characters of CODED-ID are interpreted as seven hex digits which, when non-zero, index SECTION-TABLE. Each indexed character in SECTION-TABLE is moved to ID-SECTS.

12.2. **External Addresses:**

IDDECODE Subroutine entry point.

12.3. Called Subroutines: None.

13. PRINTL

13.1. Description:

This EDL subroutine is intended to replace COBOL's I/O routines for the system printers. This routine facilitates program control of printer selection and, for printers other than $SYSPRTR, the setting of the left margin. Each call to this routine performs either an OPEN, CLOSE, NEWPAGE, or WRITE AFTER SKIP function. The function performed is determined by the value of the first word of the first parameter passed with the COBOL call statement. Each of these call types are discussed separately below.

13.1.1. OPEN

The form of the requisite COBOL call statement to perform the OPEN function is as follows:

```
CALL "PRINTL" USING parm1
```

where parm1 has the following COBOL structure.

word1	PIC S9999 COMP SYNC VALUE -6.
word2	PIC S9999 COMP SYNC VALUE left margin. (The number of spaces to the left of print position one. This has no effect if the printer name is $SYSPRTR. The default value is 16.)
word3	PIC S9999 COMP SYNC VALUE new line one. (The line number on which the line of a NEWPAGE call is printed. The default is 4.)
string	PIC X(8) VALUE printer name. (The printer's name used during System Generation. At present $SYSPRTR addresses the IBM 4974 printer and $SYS4975 addresses the IBM 4975 printer. The default value is $SYS4975.)

All subsequent calls to PRINTL will address the above referenced printer until another OPEN call is made. The calling program may switch between printers at any time by simply issuing the proper OPEN calls. However, only the initial OPEN call and those that follow a CLOSE call effect the left margin for the open printer. If the first call to PRINTL is not an OPEN request, then the default values given above apply.

13.1.2. CLOSE

The form of the requisite COBOL call statement to perform the CLOSE function is as follows:

```
CALL "PRINTL" USING parm1
```

where parm1 has the following COBOL structure.

```
word1   PIC S9999 COMP SYNC VALUE -4.
```

This call is optional. Its primary purpose is to logically close the current spool job. The effect of this call is automatic at end-of-job and, therefore, not required. It would only be used if it was desired to end a spool job, change the left margin on a subsequent OPEN call to the same printer, and then generate a new report.

13.1.3. NEWPAGE

The form of the requisite COBOL call statement to perform the NEWPAGE function is as follows:

```
CALL "PRINTL" USING parm1 parm2 ... parmn
```

where the n parms have the following COBOL structures.

parm1:

```
word1   PIC S9999 COMP SYNC  VALUE -2.

word2   PIC S9999 COMP SYNC  VALUE string length.

string  PIC X(string length) VALUE string to be printed.
```

parmi, i = 2 through n-1 (optional):

```
word1   PIC S9999 COMP SYNC  VALUE number of times to overprint
        (1-7).

word2   PIC S9999 COMP SYNC  VALUE string length.

string  PIC X(string length) VALUE string to be overprinted.
```

parmn (required):

```
word1   PIC S9999 COMP SYNC VALUE 0.
```

The NEWPAGE call causes the string in parm1 to be printed on the printer at the line defined in the most recent OPEN call. Then the routine steps through the parameter string, processing each successive parameter as an overprint parameter, until a parameter is found whose overprint value is zero. Thus, all NEWPAGE calls must have at least two parameters. Each overprint parameter causes the overprint parameter string to be printed on the current line the number of times given in the overprint count. The overprint count is formed by ANDing the value in word1 with 7. It is the calling program's responsibility to maintain the current line number and force new pages appropriately. No automatic pagination is provided by this routine.

13.1.4. WRITE AFTER SKIP

The form of the requisite COBOL call statement to perform the WRITE AFTER SKIP function is as follows:

```
CALL "PRINTL" USING parm1 parm2 ... parmn
```

where the n parms have the following COBOL structures.

parm1:

```
word1    PIC S9999 COMP SYNC  VALUE >=0

word2    PIC S9999 COMP SYNC  VALUE string length.

string   PIC X(string length) VALUE string to be printed.
```

parmi, i = 2 through n-1 (optional):

```
word1    PIC S9999 COMP SYNC  VALUE number of times to overprint
         (1-7).

word2    PIC S9999 COMP SYNC  VALUE string length.

string   PIC X(string length) VALUE string to be overprinted.
```

parmn (required):

```
word1    PIC S9999 COMP SYNC VALUE 0.
```

The WRITE AFTER SKIP call causes the string in parm1 to be printed on the next line after skiping the number of lines given in word1 of parm1. Then the routine steps through the parameter string, processing each successive parameter as an overprint parameter, until a parameter is found whose overprint value is zero. Thus, all WRITE AFTER SKIP calls must have at least two parameters. Each overprint parameter causes the overprint parameter string to be printed on the current line the number of times given in the overprint count. The overprint count is formed by ANDing the value in word1 with 7.

13.2. External Addresses:

PRINTL	Subroutine entry point.

13.3. Called Subroutines: None.

14. PUTALL

14.1. Description:

This EDL subroutine, intended to be called from a COBOL program, translates character coded information, packed one character per byte, into hex coded information, packed two hex digits per byte. The primary use of this subroutine is to pack MCCRES scores into the 80 character area in the MCCRES Evaluations Master File format from a 160 character area used for processing by a COBOL program. The actual translation performed is based upon a 16 byte translate table which provides the translated character for each of the possible hex values.

The form of the requisite COBOL call statement is as follows:

```
CALL "GETALL" USING dest,trans,source,index
```

where the four arguments are defined as follows.

dest	The hex coded destination area which must be at least (L+1)/2 bytes long where L is the value of the fourth argument.
trans	The 16 byte translate table.
source	The character source area which must be at least L bytes long where L is the value of the fourth argument.
index	The index (PIC S9999 COMP SYNC) of the last character to be translated. Presently this must have a value of from 1 to 160, inclusive.

14.2. External Addresses:

PUTALL	Subroutine entry point.

14.3. Called Subroutines: None.

15. PUTONE

15.1. Description:

This EDL subroutine, intended to be called from a COBOL program, translates a character code to a hex coded digit. The primary use of this subroutine is to pack a MCCRES score into the 80 character area in the MCCRES Evaluations Master File format from a character code processed by a COBOL program. The actual translation performed is based upon a 16 byte translate table which provides the translated character for each of the possible hex values.

The form of the requisite COBOL call statement is as follows:

```
CALL "GETONE" USING dest,trans,source,index
```

where the four arguments are defined as follows.

dest	The hex coded destination area which must be at least (L+1)/2 bytes long where L is the value of the fourth argument.
trans	The 16 byte translate table.
source	The character source byte.
index	The index (PIC 9999 COMP SYNC) of the hex code to which the translated character is to be placed. Presently this must have a value of from 1 to 160, inclusive.

15.2. External Addresses:

PUTONE	Subroutine entry point.

15.3. Called Suoroutines: None.

16. ROMNUM

16.1. Description:

This COBOL subroutine, intended to be called from a COBOL program, will convert an unsigned four digit decimal number to a left justified Roman Numeral character string. The range of this conversion is from one (I) to 3999 (MMMCMXCIX). The limit of 3999 is imposed because the Roman Numeral Characters for 5000 and 10000, V bar and X bar, respectively, are not included in the normal character sets on the system printers. The routine will operate to 9999 substituting W for V bar and Y for X bar. This extension was included in lieu of a special error routine. The decimal number must be strictly numeric including leading zeros. If this is not the case, the routine returns a null character string as an error indicator.

The form of the requisite COBOL call statement is as follows:

```
CALL "ROMNUM" USING parm1
```

where parm1 has the following COBOL structure.

WORD1	PIC S9999 COMP SYNC. (The supplied value is ignored. This routine returns the length of the returned Roman Numeral character string here. If the next field is not strictly numeric this value is set to zero.)
number	PIC 9999 VALUE number to be converted.
string	PIC X(16). (The supplied contents are ignored. The converted Roman Numeral character string is placed here left justified. If the above field is not strictly numeric this field as set to <u>all</u> blanks.)

16.2. External Addresses:

ROMNUM	Subroutine entry point.

16.3. Called Subroutines: None.

17. SLIDET

17.1. Description:

This EDL subroutine, intended to be called by a COBOL program, left or right justifies words in a character string. The subroutine has two entry points, SLIDEL to left justify and SLIDER to right justify. The supplied character string is considered to contain words, each separated by one or more blanks. The returned character string (the same area of memory) will contain the same words, each separated by a single blank, with the first word beginning at the first position for SLIDEL or the last word ending in the last position for SLIDER. If the supplied character string is all blank, it is returned all blank.

The form of the requisite COBOL call statement is as follows:

```
CALL "SLIDEi" USING parm1
```

where i is either L or R and parm1 has the following COBOL structure.

word1	PIC S9999 COMP SYNC VALUE maximum string length. (Not used by this subroutine.)
word2	PIC S9999 COMP SYNC VALUE current string length.
string	PIC X(maximum string length) VALUE supplied words.

17.2. External Addresses:

SLIDEL	Subroutine entry point.
SLIDER	Subroutine entry point.

17.3. Called Subroutines: None.

18. SQROOT

18.1. Description:

This EDL subroutine, intended to be called by a COBOL program, computes the square root of a non-negative fixed point number using an iterative algorithm, namely Newton's Method. Simply stated, to compute the square root r of S with an error less than e we define

$$r(0) = k$$

where k is any constant and successively compute

$$r(i) = \frac{r(i-1) + S/r(i-1)}{2}$$

varying i from 1 to n such that

$$|r(n) - r(n-1)| < e$$

The form of the requisite COBOL call statement is as follows:

```
CALL "SQROOT" USING sqroot-work
```

where sqroot-work has the following COBOL structure.

```
1   sqroot-work.
 2  NUMBER        PIC S9(i)V9(f) COMP SYNC.
 2  SCALE         PIC S9999      COMP SYNC VALUE f.
```

In the above i and f must both be non-negative and their sum must be less than 10.

SQROOT replaces NUMBER with its truncated square root. Note that the square root has the same precision as NUMBER. Thus, the square root of 2 is computed to be as follows:

f	square root	iterations
0	1	7
8	1.41421356	6

Throughout testing the number of iterations required varied from five to seven.

18.2. External Addresses:

SQROOT	Subroutine entry point.
ITERCNT	Iteration Count (word).

18.3. Called Subroutines: None.

19. TRANS1

19.1. Description:

This EDL subroutine will translate a character string, defined as a TEXT statement, to the index of it's first occurrence in a table of TEXT statements. Further, if requested, TRANS1 will then translate this index to the value of the corresponding entry in a table of single precision words. This translate table may be a table of addresses.

The format of the requisite EDL call statement is as follows:

```
    CALL      TRANS1,arg,tab,reti,itab
```

where the four arguments are addresses defined as follows.

arg	The address of a TEXT statement containing the character string argument to be translated. The current length is the argument's length.
tab	The address of the first TEXT statement in a table of TEXT statements to be searched for a match with the argument. For each table TEXT entry, the maximum and current lengths are taken to be the maximum and minimum compare lengths, respectively. A match occurs for the first table entry which, along with the argument, satisfies three conditions as follows:

1) table entry minimum compare length not greater than the argument's length,

2) table entry maximum compare length not less than the argument's length, and

3) table entry and argument equal for a compare length equal the argument's length.

A not found condition results in an untranslated index of zero. The end of table condition is indicated by a table entry of zero maximum length. The following is a typical table of three entries requiring an exact match for a found result.

```
TABLE    TEXT  'FIRST ENTRY'
         TEXT  'SECOND ENTRY'
         TEXT  'THIRD ENTRY'
         DC    X'00'             END OF TABLE INDICATOR
```

A similar table requiring equivalence on the first word only of each table entry would be as follows:

```
               ALIGN
               DC     X'0B05'
       TABLE   DC     'FIRST ENTRY'
               ALIGN
               DC     X'0C06'
               DC     'SECOND ENTRY'
               ALIGN
               DC     X'0B05'
               DC     'THIRD ENTRY'
               ALIGN
               DC     X'0000'          END OF TABLE INDICATOR
```

reti — The address of the index to be set indicating the results of the execution of this subroutine. This index is a single precision integer.

itab — The address of a table of single precision integers which are the corresponding translated indexes to be returned for the entries in the 'tab' table of TEXT statements. This value is zero when the index 'reti' is not to be translated. Note: this address locates the translation value for the first entry in the 'tab' table and it must be preceeded by A value for the translation of the not found condition if such a condition may arise in the execution of the subroutine. A typical table for either of the above tables follows.

```
               DC     A(NFRTN)       NOT FOUND ROUTINE
       ITAB    DC     A(FERTN)       FIRST ENTRY ROUTINE
               DC     A(SERTN)       SECOND ENTRY ROUTINE
               DC     A(TERTN)       THIRD ENTRY ROUTINE
```

19.2. External Addresses:

TRANS1 — Subroutine entry point.

19.3. Called Subroutines: None.

20. UPCASE

20.1. Description:

This EDL subroutine is non-executable. It is simply a 256 byte table in which each byte contains the binary value representing its location in the table with the exception of those positions corresponding to the lower case alphabetic characters each of which are replaced by the upper case value. This table is normally used to translate lower case characters to upper case characters.

20.2. External Addresses:

UPCASE	Table Location.
UPCASE#	Table Location.

20.3. Called Subroutines: None.

21. UNIQUE

21.1. Description:

This EDL subroutine, intended to be called from a COBOL program, will left justify an ordered set of unique characters taken from the original contents of a supplied character string. The resultant length of the returned character string is also returned.

The form of the requisite COBOL call statement is as follows:

```
CALL "UNIQUE" USING parm1
```

where parm1 has the following COBOL structure.

word1	PIC S9999 COMP SYNC VALUE maximum string length. (Not used by this subroutine.)
word2	PIC S9999 COMP SYNC VALUE current string length. (Upon entry this word contains the length of the string to be scanned for unique characters. Upon exit this word contains the number of unique non-blank characters found.
string	PIC X(maximum string length) VALUE source/destination string. (Upon entry this string contains the characters to be scanned. Upon exit this string begins with the ordered unique set of characters found. The positions to the right of the last unique character contain the original contents of this string.)

21.2. External Addresses:

UNIQUE	Subroutine entry point.

21.3. Called Subroutines: None.

22. UNITI2

22.1. Description:

This COBOL subroutine delivers a MCCRES Unit File record identified by its MCCRES Volume and historical evaluation number. It is required that each unit file be named Ui where i is a MCCRES volume number, that only the named MCCRES Volume be represented in the file, and that the file be in strict ascending sequence on historical evaluation number. Reference to the file is by volume and evaluation numbers directly on a 'best guess' basis and then sequentially, either forward or backward to locate the requested record. If the requested record does not exist a dummy record containing only the MCCRES Volume number and historical evaluation number is constructed and given a logical record number of zero.

The format of the requesite COBOL call statement is as follows.

```
CALL "UNITI2" USING unit-parm log-rcd-area.
```

where the parameters have the following minimum COBOL definitions.

```
1   unit-parm.
 2  UP-ERR-CODES       PIC S9(9) COMP SYNC.
 2  UP-ERR-NUM         PIC S9999 COMP SYNC.
 2  UP-VOL             PIC 99.
 2  UP-EVAL            PIC 999.
 2  UP-EDXVOL          PIC X(6).
*
1   log-rcd-area.
 2  LOG-RCD-NUM        PIC S9(9) COMP SYNC.
 2  LOG-RCD.
  3   UNIT-VOL         PIC 99.
  3   FILLER           PIC X.
  3   UNIT-EVAL        PIC 999.
  3   FILLER           PIC X(122).
```

The calling program must have filled in UP-VOL, UP-EVAL, and UP-EDXVOL (the EDX volume name where the unit files are located, typically HRDMCV). Upon return from this routine, UP-ERR-NUM will be unchanged for a successful request, or be set to 2 if the error occurred during the 'best guess' routine, to 3 if the error occurred during the 'get-next' routine, or to 4 if the error occurred during the 'get-prev' routine. In any event, the error is identified by UP-ERR-CODES (see Section 6.1.6).

22.2. External Addresses:

UNITI2   Subroutine entry point.

22.3. Called Subroutines: GETRCD.

23. UNITID

23.1. Description:

This EDL subroutine, intended to be called by a COBOL program, delivers a MCCRES Unit File record identified by its MCCRES Volume and historical evaluation number. The operation of this routine is similar to that of its successor, UNITI2, described earlier. In fact, much of the discussion of Section 22.1 applies.

The format of the requisite COBOL call statement is as follows.

```
CALL "UNITID" USING unit-parm.
```

where unit-parm has the following minimum COBOL definition.

```
1  unit-parm.
 2  UP-EDXVOL          PIC X(6).
 2  UP-VOL             PIC 99.
 2  UP-EVAL            PIC 999.
 2  UP-FOUND-SW        PIC X.
 2  UP-EOF-SW          PIC X.
 2  FILLER             PIC X.
 2  UP-RCD-NUM         PIC S9999 COMP SYNC.
 2  UNIT-RCD.
  3   UR-VOL           PIC 99.
  3   FILLER           PIC X.
  3   UR-EVAL          PIC 999.
  3   FILLER           PIC X.
  3   UR-TEXT          PIC X(20).
  3   FILLER           PIC X.
  3   UR-DATE          PIC X(8).
  3   FILLER           PIC X.
  3   UR-LIST          PIC X(2).
  3   FILLER           PIC X.
  3   UR-POR           PIC X(4).
  3   FILLER           PIC X(9).
```

The calling routine must set UP-EDXVOL to the EDX volume label where the unit files are located and UP-VOL and UP-EVAL to the MCCRES Evaluation for which the unit record is requested. This routine sets UP-FOUND-SW, UP-EOF-SW, AND UP-RCD-NUM appropriately. UR-VOL AND UR-EVAL are set whether or not the requested unit record is found. If the requested unit record is not found, the remainder of UNIT-RCD is blanked.

23.2. External Addresses:

UNITID	Subroutine entry point.

23.3. Called Subroutines:

24. YNRESP

24.1. Description:

This EDL subroutine will request a YES or NO response from the user and return to the calling routine according to the user's response. The calling routine has control over whether a stored (previous) user response is acceptable or not, whether the response will be requested on the prompt line or the next line, and if a prompt is given at all. Upper and lower case characters are equivalent in any user response (each character is converted to upper case before processing) and any leading portion of the words YES and NO are equivalent to full words. Thus, a user may respond either 'y' or 'n' to effect YES or NO, respectively. Invalid responses result in a diagnostic message being displayed and this routine being repeated with the force user response switch being set if not already set. This process will repeat indefinitly until the user supplies a valid response.

The format of the requisite EDL call statement is as follows:

```
   CALL     YNRESP,msgadr,yesrtn,nortn,ctlwrd
```

where the four arguments are defined as follows.

msgadr	The address of a TEXT statement containing the prompting character string. If no prompt is desired this argument must be zero. Weather or not the calling program provides a prompt, this routine sounds the bell requesting the user's attention.
yesrtn	The address of the routine to be entered when the user's response is YES. If it is desired to have the statement following the subroutine call executed for a YES response this operand must be zero.
nortn	The address of the routine to be entered when the user's response is NO. If it is desired to have the statement following the subroutine call executed for a NO response this operand must be zero.
ctlwrd	This argument is a control word which is passed on to the GETWRD subroutine to control the retrieval of the user's response. For information on the bit settings and their meanings refer to GETWRD for a full explanation. However, only the bits X'10' (force a user's response) and X'01' (user response on the following line) are allowed by this subroutine. All other bit positions are forced to zero upon entry to this subroutine.

If it is desired to retrieve the capitalized user response it may be accessed via the external tag YN$WORK which addresses a four character TEXT statement. This might be desired when the user response controls many branches throughout the program and it is desired to save a user's response without altering the program flow, i.e., both the yesrtn and nortn arguments are zero.

24.2. External Addresses:

YNRESP	Subroutine entry point.
YN$WORK	Four character user response TEXT area - upon return character one will be either a Y or N.

24.3. Called Subroutines: GETWRD, TRANS1, & UPCASE.

REFERENCES

[1] BARZILY, Z. (1980). Analyzing MCCRES data. Technical Paper Serial T-427.

[2] BARZILY, Z., P. R. CATALOGNE, and W. H. MARLOW (1981). Assessing Marine Corps readiness. Defense Management Journal, Vol. 18, No. 1, pp. 25-29.

[3] BRIER, S. S., S. ZACKS, and W. H. MARLOW (1985a). An application of empirical Bayes techniques to the simultaneous estimation of many probabilities. Technical Paper T-486. To appear in Naval Research Logistics Quarterly.

[4] BRIER, S. S., S. ZACKS, and W. H. MARLOW (1985b). Results of a simulation study to measure the effectiveness of empirical Bayes' estimates of multiple probabilities. Technical Paper T-499 (forthcoming).

[5] CAVES, W. E. (1985a). Summary of the GWU Marine Corps Combat Readiness Evaluation System Software Applications (MCCRESSA). Technical Paper T-498.

[6] CAVES, W. E. (1985c). Marine Corps Combat Readiness Evaluation System Software Applications (MCCRESSA) support programs. Technical Paper T-502.

[7] CAVES, W. E. and W. H. MARLOW (1985). Marine Corps Combat Readiness Evaluation System (MCCRES) data base. Technical Paper T-503 (forthcoming).

[8] ZACKS, S. and W. H. MARLOW (1982). Estimating the structural parameters of the Marine Corps Combat Readiness Evaluation System on the basis of the primary categories model. Technical Memorandum Serial TM-69200.

[9] ZACKS, S., W. H. MARLOW, and Z. BARZILY (1981). Category analysis of the Marine Corps Combat Readiness Evaluation System. Technical Paper T-450.

[10] ZACKS, S., W. H. MARLOW, and S. S. BRIER (1985). Statistical analysis of very high-dimensional data sets of hierarchically structured binary variables with missing data: an application to Marine Corps readiness evaluations. Naval Research Logistics Quarterly, Vol. 32, pp. 467-490.